# SBMLmerge, a system for combining biochemical network models

**Marvin Schulz**[*] **Jannis Uhlendorf**[*]
`schulzma@molgen.mpg.de` `uhlndorf@molgen.mpg.de`
**Edda Klipp** **Wolfram Liebermeister**[†]
`klipp@molgen.mpg.de` `lieberme@molgen.mpg.de`

Max Planck Institute for Molecular Genetics, Ihnestraße 63-73, 14195 Berlin, Germany
[*] These authors have contributed equally to this work
[†] Corresponding author

**Abstract**

The Systems Biology Markup Language is an XML-based format for representing mathematical models of biochemical reaction networks, and it is likely to become a main standard in the systems biology community. As published mathematical models in cell biology are growing in number and size, modular modeling approaches will gain additional importance.

The software SBMLmerge assists the user in combining models of biological subsystems to larger biochemical networks. First, the programs helps the user in annotating all model elements with unique identifiers, pointing to databases such as KEGG or Gene Ontology. Second, during merging, SBMLmerge detects and resolves various syntactic and semantic problems. Typical problems are conflicting variable names, elements which appear in more than one input model, but also mathematical problems arising from the combination of equations. If the input models make contradicting statements about a biochemical quantity, the user is asked to choose between them. In the end, the merging process results in a new, valid SBML model.

**Keywords:** Model combination, SBML, MIRIAM, Metabolic model

## 1 Introduction

The molecular processes in cells form a huge network, which makes detailed mathematical modeling and simulation of cells extremely difficult. A potential strategy to obtain large models is to construct them “bottom-up” from smaller modules that can be constructed and understood more easily. Snoep *et al.* demonstrated this approach by combining a model of the glycolysis with a model of the glyoxylate pathway [6]. A growing number of models is publicly available in databases such as Biomodels [16] or JWS [9]. For the future, we expect that more and more biochemical models will be constructed out of smaller models. This requires both (i) standards for the representation of biochemical models (e.g. SBML [1], CellML [4]) and (ii) standardised experimental conditions. In addition, modellers will also need efficient algorithms for model merging and for the detection of problems during the merging process.

We developed the software SBMLmerge to facilitate the bottom-up modelling approach, providing a general method for the combination of biochemical networks given in SBML format (see below). The models to be integrated can contain biologically identical compounds or reactions. Out of these identical elements, only one representative is retained in the output model. This is illustrated in figure 1: the first reactions of glycolysis and glycogen production, modelled with arbitrary enzyme kinetics, are merged by SBMLmerge. As requested, the common model elements appear only once in the output model.

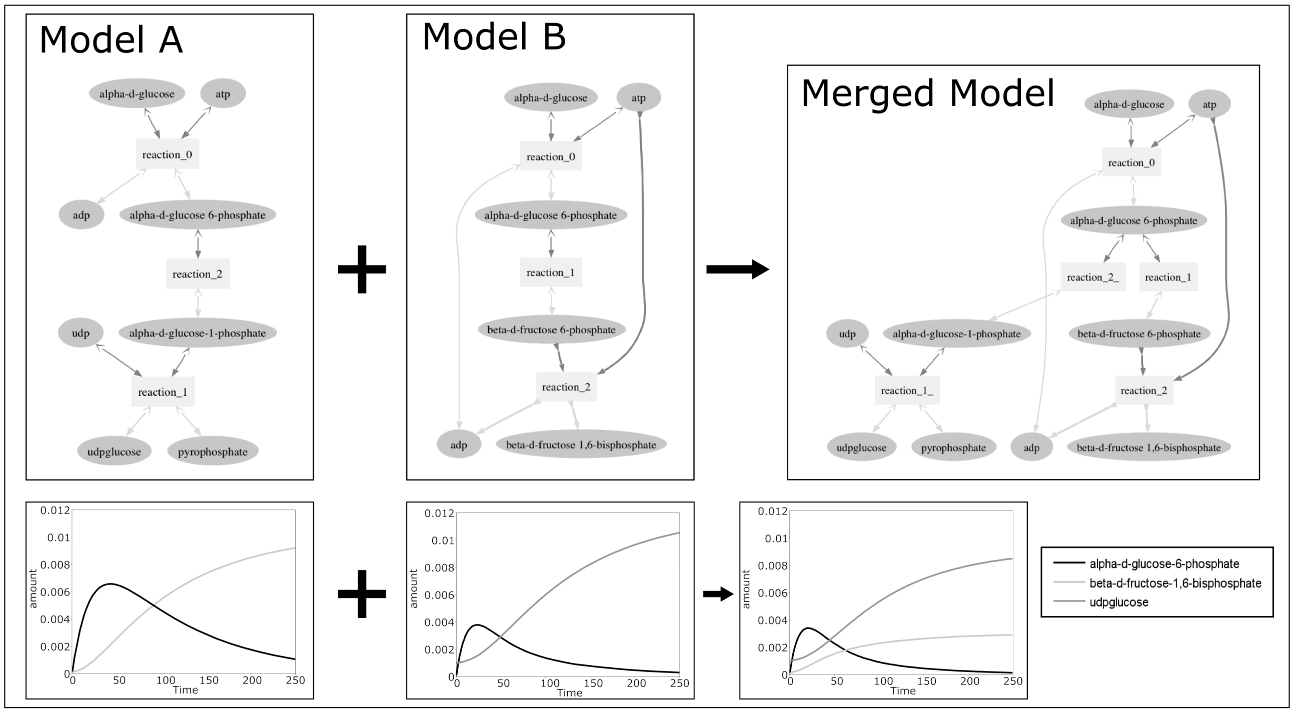
Model A
alpha-d-glucose
atp
reaction_0
adp
alpha-d-glucose 6-phosphate
reaction_2
udp
alpha-d-glucose-1-phosphate
reaction_1
udpglucose
pyrophosphate
Model B
beta-d-fructose 6-phosphate
beta-d-fructose 1,6-bisphosphate
Merged Model
reaction_2_
reaction_1_
amount
Time
alpha-d-glucose-6-phosphate
beta-d-fructose-1,6-bisphosphate
udpglucose

Figure 1: Merging of two small metabolic models. Both models represent the production of glucose-6-phosphate from glucose, followed by the first steps of glycogen production and glycolysis, respectively. From these two models, SBMLmerge produces a merged output model in which the common chemical reaction shows up only once. Model topologies were drawn using SBML2dot. After the merging, all three models have been simulated with CellDesigner [7]. The result of the simulation of the merged model is plausible. Since the production of UDP-glucose in the second model is a little faster than the production of fructose-1,6-bisphosphate in the first model, it has a higher steady state concentration in the merged model. Of course the concentrations of both metabolites is lower in the end in the merged model, since the available amount of glucose is degraded to both of them.

SBML [1] is a format for representing mathematical models of biochemical networks. It is based on XML [23], and we expect it to become a main standard for the exchange of such models. SBML describes biological processes independently of the type of mathematical description or analysis. Therefore, its structure represents the topology of the biochemical network, rather than differential equations for metabolite concentrations. An SBML file declares a set of metabolites (”species”), which are connected by a set of chemical reactions. A mathematical expression for the velocity of each reaction can be specified in the MathML [22] format. If a metabolite is not modified by any reaction, its concentration can also be described by differential or algebraic equations (”rules”). Definitions that should only be evaluated in response to a certain trigger can be modelled in SBML by *events*. Furthermore it is possible to define different compartments which can contain different concentrations of the same metabolites.

## 2 Challenges when merging SBML files

Merging can only lead to useful models if all input models are correct. There exist a variety of problems that can make a model invalid. For example, the model may contradict the biological reality due to incorrect data which the model is based on, or wrong assumptions, for example reactions that do not exist in a certain organism. Also mathematical problems can arise, like under-/overdetermined equation systems, or algebraic equations which cannot be solved consecutively because they contain

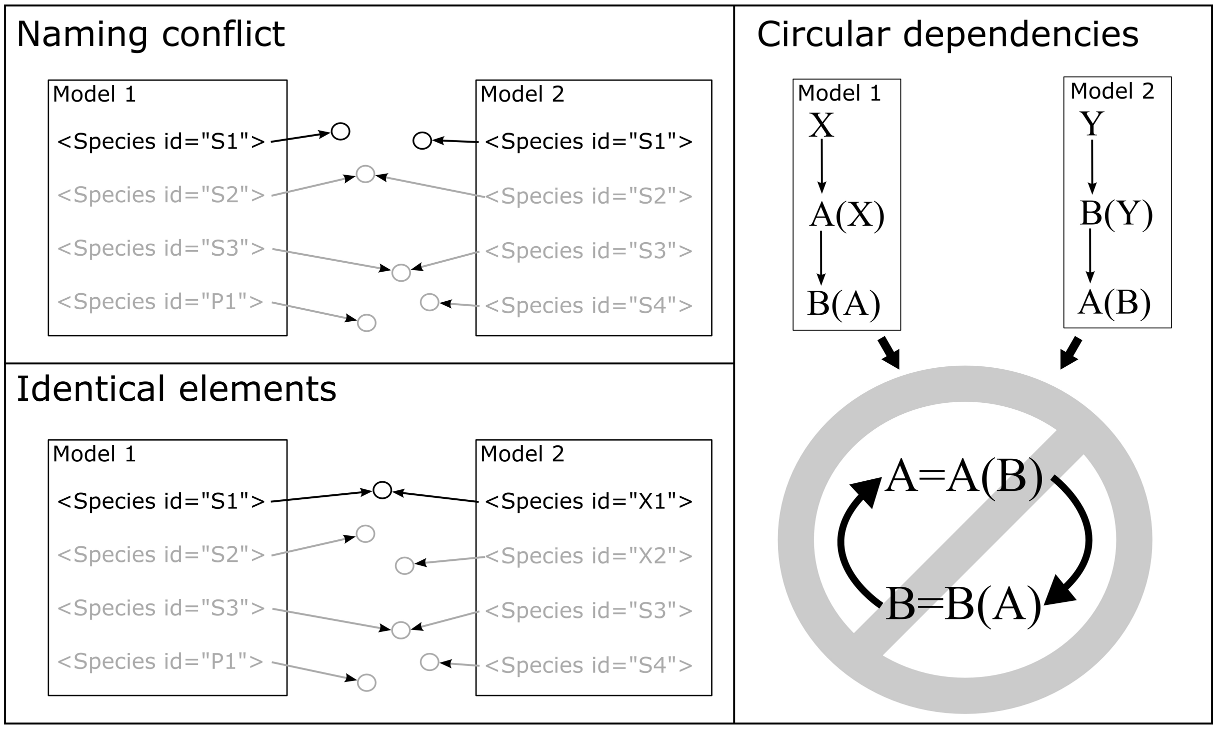

Figure 2: Problems that can arise during merging of biochemical models. Top left: In the two models, two distinct elements carry the same name. Bottom left: Two identical elements carry different names. Right: Loops of algebraic equations. In model 1, variable A is computed from variable B, and in model 2, B depends on A. Combining these two equations would result in a loop that leaves both variables undetermined.

circles.

But also when the input models are valid, a program to combine them automatically still has to deal with various problems, some of which are illustrated in figure 2. These problems can be divided into two different classes. First, there are *syntactical requirements* due to the structure of SBML, like the uniqueness of IDs within a model or the restriction that a variable cannot have multiple assignments. Of course, two models can both contain a species with the ID "A", but if these models are merged, one of these species has to be either omitted or renamed in the output file.

In addition, there are *semantical problems* like the identification of biologically identical elements. For example, the merging process has to identify species that represent identical substances. Since identical substances with identical physical locations can appear only once in a model, such multiple references have to be replaced by a single SBML species during the merging process. Also, the program has to detect biologically contradicting elements, such as *overlapping* compartments. For instance, "cell" and "mitochondrion" are overlapping, since the mitochondrion is a part of the cell. Overlapping compartments must be avoided since the corresponding metabolite concentrations would be dependent variables. This is especially undesirable if the variables have been inherited from different models, because then there would be no information on their mathematical relation.

Another kind of problems are loops of algebraic equations, which resemble *circular definitions*. For example, if one model states that the amount of $ATP$ depends on the amount of $ADP$ (e.g. $ATP = 2ADP$), while the statement is adverse in the second model (e.g. $ADP = 0.5ATP$), only one of the equations can be kept, otherwise the resulting model would not be computable.

In any case, SBMLmerge needs to identify the biological meaning of all model elements in order to find identical or contradicting elements. To clarify the relation between model elements and the corresponding biological entities, we have to distinguish between different kinds of attributes in the

SBML elements.

**Element specific** attributes are used to describe the biological entity that is referred to. They can be divided into two different classes:

- **Identifying** attributes define the element's name or how it is referred to in databases.
  For example, for a species tag the ID, the metaID, the name, and the annotation attributes specify the chemical substance described.
- **Describing** attributes define the biological identity of an element.
  An important information about a reaction is its set of participating species. This information is crucial for the function of a reaction, but it still cannot be called *identifying*. For example, there might be reactions, catalysed by unmentioned isoenzymes, which have the same sets of reactants and products, but still have to be regarded as different.

**Model specific** attributes contain information that is only meaningful in the context of the model, or statements that are made by the model.
For example, the *constant* attribute of a species tells whether the concentration of a species is constant or variable over time.

The *identifying* attributes are used to search models for elements that refer to the same biological entity. Two elements which have the same value for any of these *identifying* attribute are called *conflicting*. If *conflicting* elements also share common values for their *describing* attributes, they are called *identical*, but if these values differ, they are called *contradicting*.

We have already seen that model annotation and model merging can be treated separately. Altogether, the SBMLmerge program consists of several subroutines: **SBMLannotate** assists the user in specifying the biological meaning of his model elements, **SBMLcheck** performs various consistency checks on the syntactical and semantical validity of an SBML file, **SBMLmerge** helps the user in consolidating the mathematical and biological information from several models into one single, consistent SBML file, and **SBML2dot** draws a graphical representation of the output model. The subprograms can be executed individually within other applications. In the following sections, we shall have a closer look at each of them.

## 3 SBMLannotate

Model merging requires that the biological meaning of the model elements has been declared. For this reason, a necessary step before model merging is to annotate SBML elements, that is, to link them to IDs from databases explaining their biological meaning. SBMLannotate suggests annotations automatically or lets the user navigate through an SBML file to annotate the compartments, species, and reactions separately. For each element, SBMLannotate suggests related database IDs to the user, either based on the XML code of this element or based on search strings provided by the user. To speed up the search for IDs, a local database is used instead of web-services.

Since gathered biological information should be reusable, it has to be stored somewhere inside the SBML code. For this task we have chosen SBML's *annotation* attribute, which is a string in XML format. A standard of how links to databases are stored inside the SBML code has been developed by the MIRIAM [3] initiative. This standard allows for different kinds of relations between model elements and database entries. These relations, which have been taken from Dublin Core [8], are given in table 1. Currently only the *relation* relation, which indicates that the SBML element has the specific function described in the database, is supported by SBMLmerge.

Figure 3 shows an example for a species that has been assigned a database ID in a MIRIAM compliant annotation, using RDF [21] and Dublin Core.

| DC term | Use cases for species... |
|---|---|
| *relation* | The substance in the database is the one described by the SBML species. |
| *isPartOf* | The SBML species is a part of the complex species explained in the database. |
| *hasParts* | The SBML species is a complex, which consists of parts that are referred to by these database entries. |
| *isVersionOf* | The SBML species is one substance out of the set of substances referred to in the database. |
| *hasVersion* | The SBML species represents a set of substances which are described by the database entries. |
| *isReferencedBy* | The SBML species is referenced by the given resource |

Table 1: Dublin Core relations used in the MIRIAM format. Currently only the DC term *relation* is supported by SBMLmerge.

## 3.1 SBMLannotate ID database

The SBMLannotate ID database relates *identifying* SBML content to IDs from various databases. Its structure is fairly simple: for the conversion of species into IDs, it contains a list of substrate names and their links to the databases Gene Ontology [18], KEGG Compound [19], ChEBI [17], PubChem [10], 3DMET [14] and CAS [15]. This information has mainly been extracted from Gene Ontology and KEGG, which contain cross-linking information to other databases. For the conversion from compartments into IDs, the database contains a list of names for physical compartments (like mitochondrion, cytoplasm, etc.) mapped to IDs from Gene Ontology. For chemical reactions, it contains a list of KEGG Reaction IDs together with the KEGG Compound IDs of all substances that participate in each reaction. This information is used by SBMLannotate to assign annotations to the compartments, species, and reactions from a given SBML model.

## 3.2 Automatic suggestion of IDs

SBMLannotate contains a subroutine which suggests database IDs for given SBML elements. First, identifying information from the SBML file is gathered. In the case of compartments or species, the identifying attributes of these element are searched for potential names. In the case of reactions, KEGG Compound IDs for the educts and products are looked up in the SBML code. Afterwards, this information is used to search for potential IDs in the database: In the case of compartments or species, potential names are simply compared with the list of known names. Every gathered potential name that appears in this list is suggested to the user together with its associated database IDs. For reactions, the database search works differently. Here the user is presented a list of all reactions that share all educts and products from the SBML code (as well as, possibly, other substances).

## 3.3 Fuzzy database search

Besides the automated search for names inside the SBML code, the user can also type in presumable names for compartments and species. SBMLannotate uses a fuzzy matching algorithm to look up these names in the ID database and to suggest the most similar names. The algorithm ("Gestalt Pattern Matching") is explained in figure 4. Despite its simplicity, the algorithm works well in practice since the database contains many alternative substance names. For example, it contains the names "2-Deoxy-D-arabino-hexose" and "D-arabino-2-Deoxyhexose", which both correspond to the KEGG Compound "C00586".

```
<species id="s1" metaid="s1" name="glucose-6-phosphate" compartment="cytosol">
  <annotation>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
             xmlns:dc="http://purl.org/dc/elements/1.1/">
      <rdf:Description rdf:about="#s1">
        <dc:relation>
          <rdf:Bag>
            <rdf:li rdf:resource="http://www.genome.jp/kegg/compound/#C00092" />
          </rdf:Bag>
        </dc:relation>
      </rdf:Description>
    </rdf:RDF>
  </annotation>
</species>
```

Figure 3: SBML species with a MIRIAM compliant annotation. The annotation points to the entry C00092 in the KEGG Compounds database, which represents glucose-6-phosphate. To create such an annotation, SBMLannotate suggests a list of database IDs, based on information extracted from the SBML code. For this species, SBMLannotate would have suggested the ID "C00092" from the KEGG Compound database, as the name "glucose-6-phosphate" is listed together with this ID in the SBMLmerge ID database. After the user has selected this ID, the annotation is inserted into the SBML element.

## 4 SBMLcheck

SBMLcheck checks SBML models for semantic problems and reports them to the user. Its purpose is to ensure that only correct models are used as input for SBMLmerge. Combining correct models with SBMLmerge will lead either to a correct output model, or if the merging process fails for some reason, to no output at all. Hence to ensure correctness of the output model, it is crucial to ensure the validity of the input models.

In general, the definition of validity depends on the purpose of a model. For example, a model formulated in terms of chemical reactions has other requirements for being valid than a model defined by differential equations. SBMLcheck checks the model for requirements that are universal for different ways of modelling and that are also relevant for model combination. Models can be invalid for syntactical or semantical reasons. As tools are already available to check for syntactical problems [12], SBMLcheck focuses mainly on detecting semantical problems within a model.

Some of the problems that can lead to an invalid SBML model have been discussed above. Further problems can arise if the model is incomplete, that is, if certain metabolites or reactions are missing.

SBMLcheck performs the following checks.

1. *Syntax check*
   When a file is processed, it is first checked for correct SBML syntax.

2. *Check for correct annotations*
   Like SBMLmerge, SBMLcheck requires correctly annotated models. SBMLcheck searches for elements that are not annotated and displays them to the user. Correct annotations are necessary for some of the following checks.

3. *Mathematical rule check*
   According to the SBML specification, all variables that appear in an assignment rule must have been specified by a preceding rule. This excludes loops between algebraic rules. SBMLcheck verifies that the rules can be evaluated consecutively in the order they are given.

First, the longest common substring is found.

```
  Glucose-6-phosphate
            |||||||||
Glucose six phosphate
```

Then the string is divided on both sides of this subsequence and the algorithm is recursively executed on the left and the right side.

```
Glucose-6-  phosphate
|||||||     |||||||||
Glucose six phosphate
```

Finally, the matches, mismatches, and overlaps are scored.

```
Glucose-6-**phosphate
|||||||---**|||||||||
Glucose six phosphate
```

Figure 4: Gestalt pattern matching. The algorithm developed by Ratcliff and Obershelp [5] compares two different strings by recursively finding their longest common subsequence. It is implemented in Python [11] in the module *difflib*. In this example, the algorithm is used to compare the strings "Glucose-6-phosphate" and "Glucose six phosphate".

4. *Overlapping annotation check*
   SBMLcheck checks all elements for their annotations and warns the user if two elements carry the same annotation, which would be a sign of overlapping elements. To check for overlapping compartments, SBMLcheck uses the Gene Ontology, which represents spatial relations between cell compartments as a directed acyclic graph (DAG). For example, this DAG indicates that the mitochondrial matrix is situated inside the mitochondrial lumen, which again is part of the mitochondrion. To use Gene Ontology, it is necessary that all compartments are annotated with Gene Ontology identifiers.

5. *Reaction balance check*
   SBMLcheck investigates the balance of atom numbers in reactions. In a correct reaction, the number of atoms of each atom type has to be the same on both sides of the reaction formula. A change in atom numbers indicates that (i) either some metabolites (e.g., water) have been left out or (ii) an error has occurred during the modelling of the reaction. SBMLcheck counts the number of atoms of each type on both sides and raises an error if they do not match. The atomic composition of the metabolites is looked up in the KEGG database.

## 5 SBMLmerge

SBMLmerge first merges the lists of elements in the annotated input files. These lists are then searched for *conflicting* elements by pairwise comparison, based on the *identifying* attributes. If two conflicting elements are found, their *describing* attributes are compared. The values for these attributes can be identical for both conflicting elements, or they can differ in one or more values: if all attribute values are identical, then the elements are supposed to have the same biological meaning. In order to keep the models semantically valid, we have to delete one of these elements from the file. On the other hand, if the elements differ in at least one attribute, they are not considered identical, so both of them are retained in the output model.

To be more specific, we shall describe two particular conflicts that can arise during the merging process:

1. *Naming conflicts on species*
   As mentioned before, the *identifying* attributes for species are metaID, ID, name, and annotation. If two species are annotated with identical database links, their *describing* attributes

are compared. For species, the compartment attribute is *describing*, since a species cannot be defined twice in the same compartment, but in different ones. So if two species with identical annotations are found in the same compartment, they are regarded as *identical* and replaced by a single species.

2. *Variable conflicts on assignment rules*
Multiple assignments of the same variable in a single model are either redundant or contradictory and are forbidden in the SBML language. In order to find multiple assignments during merging, the *left hand sides* of assignment rules (the variable, which is changed by the rule) are regarded as *identifying* attributes. Accordingly, two rules that modify identical variables are regarded as *conflicting*. Since no further *element describing* attributes of rules are taken into account, *conflicting* assignment rules are always directly regarded as *contradicting*, and so the user has to choose one of them to be deleted from the output model.

As mentioned before, rules also have to be free of dependence circles. When lists of rules from the input models simply joined, such circles can easily arise, so SBMLmerge has to detect them and to suggest ways to remove them from the output model. If a circle is found, the user is guided in removing it by deleting preferably few rules from the ones which have caused the circle. If there have been other rules determining the same variable as the rule to be deleted, the user is also asked whether he wants to reintroduce a previously deleted rule to the output model.

A detailed list of further problems handled by SBMLmerge can be found in the user manual [13].

## 6 SBML2dot

After models have been merged, SBML2dot plots a network graph of the output model. For the actual drawing, SBML2dot calls GraphViz [2], which can draw pictures in various formats from a given plain text input file. The resulting network graph contains the species and reactions of a model and their relations to each other. The user can also choose between different colouring schemes. For example, the species and reactions can be coloured according to the model from which they were inherited. The program can also retrieve species names from the SBMLannotate database, based on the information given in the annotation tag.

## 7 Discussion

**Summary** SBMLmerge helps the user to combine several biochemical models given in SBML format. To cope with problems that occur during the merging process, we have developed SBMLannotate for attachment of unique identifiers to the elements of an SBML file, and SBMLcheck for verification of the syntactic and semantic consistency of SBML files. First, the user has to assign biological annotations to elements. These annotations enable SBMLmerge to decide whether elements describe the same or related biological quantities. The output models are valid SBML files that perform reasonably in simulations, as shown in figure 1.

The program is distributed under the GNU public license [20] and can be downloaded or accessed via a web interface at http://sysbio.molgen.mpg.de/sbmlmerge. The algorithm underlying SBMLmerge can also be used as a guideline to combine models by hand.

**Qualifying relations** At present, SBMLmerge supports only the Dublin Core term *relation* for annotation of elements. This "is a" relationship suffices to annotate simple molecules and reactions, but other relationship will be needed. To handle the mentioned problems, we plan to support other relations from Table 1 as well. For instance, a qualifying relation "has_parts" could be used for annotating complex molecules such as phosphorylated molecules, or protein complexes that consist of several subunits. Another problem for correct annotations are imprecisely specified elements, like "glucose" instead of "$\alpha$-d-glucose", or pooled elements, such as lumped metabolites or reactions. Lumped metabolites denote a sum over metabolite concentrations, for example triosephosphates instead of the two distinct molecules dihydroxyacetone phosphate and glycerine 3-phosphate. Combining models with different accuracy will cause problems, because the lumped metabolite may overlap with a precise metabolite from the other model. Likewise, lumped reactions contain different reactions that are compiled together. Conflicts between models that contain lumped metabolites or reactions could be detected based on qualifying relations in the form "has_version", as they make it possible to identify overlapping elements.

**Biological preconditions** As mentioned before, there exist several biological objections to model combination in general. Obviously, meaningful combination can only arise from models describing the same type of cell under comparable experimental conditions. From the perspective of model merging, it is vital to define uniform experimental conditions in order to ensure interoperability between the models. We are aware of the fact that this is not possible for all experiments, but we expect that further standardisation of experiments and models will be helpful for the systems biology community. While in the end, it is still up to the user to decide whether two models can be merged at all, we can conclude here that some important technical problems in model merging can be detected and solved automatically.

## Acknowledgements

The authors are supported by the German Federal Ministry for Education and Research (BMF,grant 031U109C and 0312705B); they are members of the Berlin Center for Genome based Bioinformatics, BCB. This work was funded by the European Commission, grant No. 503269.

## References

[1] Finney, A., and Hucka, M. Systems biology markup language: Level 2 and beyond, *Biochem. Soc. Trans.* 31:1472–1473, 2003.

[2] Gansner, E.R., and North, S.C. An open graph visualization system and its applications to software engineering, *Softw. Pract. Exper.* 30(11):1203–1233, 2000.

[3] Le Novère, N., Finney, A., and Hucka, M. *et al.* Minimum information requested in the annotation of biochemical models (MIRIAM), *Nat. Biotechnol.* 23:1509–1515, 2005.

[4] Lloyd, C.M., Halstead, M.D., and Nielsen, P.F. CellML: its future, present and past, *Prog. Biophys. Mol. Bio.*, 85:433–450, 2004.

[5] Ratcliff, J.W., and Metzener, D. Pattern matching: The gestalt approach. *Dr. Dobb's Journal*, 1988.

[6] Snoep, J.L., Bruggemann F., Olivier B.G., and Westerhoff, H.V. Towards building the silicon cell: A modular approach, *Biosystems* 83:207–216, 2006.

[7] http://celldesigner.org/

[8] http://dublincore.org/

[9] http://jjj.biochem.sun.ac.za/

[10] http://pubchem.ncbi.nlm.nih.gov/

[11] http://python.org/

[12] http://sbml.org/validator/

[13] http://sysbio.molgen.mpg.de/sbmlmerge/sbmlmerge_manual/

[14] http://www.3dmet.dna.affrc.go.jp/

[15] http://www.cas.org/

[16] http://www.ebi.ac.uk/biomodels/

[17] http://www.ebi.ac.uk/chebi/

[18] http://www.geneontology.org/

[19] http://www.genompe.jp/kegg/

[20] http://www.gnu.org/copyleft/gpl.html

[21] http://www.w3.org/RDF/

[22] http://www.w3.org/RT/MathML/

[23] http://www.w3.org/XML/